

Hand GESTURE robot

UPTD TIKOMDIK DISDIK JABAR

Baris Kode

Hand GESTURE robot

Baris Kode

© UPTD TIKOMDIK DISDIK JABAR

Jl. DR. Radjiman No. 6 Pasirkaliki, Kec. Cicendo Bandung

**Baris kode:**

**ArduinoIDE**

#include <Servo.h>

Servo servothumb;

Servo servoindex;

Servo servomiddle;

Servo servoring;

Servo servopinky;

char number[50];

char c;

int state = 0;

String myStringRec;

int stringCounter = 0;

bool stringCounterStart = false;

String myRevivedString;

int stringLength = 6;

int servoPinky,servoMiddle,servoIndex,servoThumb,servoRing;

int myVals[] ={0,0,0,0,0} ;

bool portable;

void setup() {

Serial.begin(9600);

servothumb.attach(9);

servoindex.attach(10);

servopinky.attach(11);

servoring.attach(12);

servomiddle.attach(13);

delay(500);

}

void loop() {

if (portable == true){

}

else{

receiveData();

if (servoPinky ==1){ servopinky.write(180);}else{servopinky.write(0);}

if (servoIndex ==1){ servoindex.write(180);}else{servoindex.write(0);}

if (servoMiddle ==1){ servomiddle.write(180);}else{servomiddle.write(0);}

if (servoThumb ==1){ servothumb.write(150);}else{servothumb.write(0);}

if (servoRing ==1){ servoring.write(180);}else{servoring.write(0);}

}

}

void allON(){

servopinky.write(180);

servoindex.write(180);

servomiddle.write(180);

servothumb.write(150);

servoring.write(180);

}

void allOFF(){

servopinky.write(0);

servoindex.write(0);

servomiddle.write(0);

servothumb.write(0);

servoring.write(0);

}

void receiveData() {

int i = 0;

while (Serial.available()) {

char c = Serial.read();

if (c == '$') {

stringCounterStart = true;

}

if (stringCounterStart == true )

{

if (stringCounter < stringLength)

{

myRevivedString = String(myRevivedString + c);

stringCounter++;

}

if (stringCounter >= stringLength) {

stringCounter = 0; stringCounterStart = false;

servoPinky = myRevivedString.substring(1, 2).toInt();

servoRing = myRevivedString.substring(2, 3).toInt();

servoMiddle = myRevivedString.substring(3, 4).toInt();

servoIndex = myRevivedString.substring(4, 5).toInt();

servoThumb = myRevivedString.substring(5, 6).toInt();

Serial.print(servoPinky);

Serial.print(" ");

myRevivedString = "";

}

}

}

}

**Python**

import cv2  
import numpy as np  
from RobotHandGestures import utlis  
##############################################################################  
  
cameraNo = 0  
portNo = "COM9"  
cropVals = 100, 100, 300, 400 # StartPointY StartPointX h w  
frameWidth = 640  
frameHeight = 480  
brightnessImage = 230  
  
##############################################################################  
  
cap = cv2.VideoCapture(cameraNo)  
cap.set(10, brightnessImage)  
cap.set(3, frameWidth)  
cap.set(4, frameHeight)  
utlis.initializeTrackBar()  
utlis.connectToRobot(portNo)  
  
while True:  
 \_, img = cap.read()  
 imgResult = img.copy()  
  
 imgBlur = cv2.GaussianBlur(img, (7, 7), 1)  
 imgHSV = cv2.cvtColor(imgBlur, cv2.COLOR\_BGR2HSV)  
 trackBarPos = utlis.getTrackbarValues()  
 imgMask, imgColorFilter = utlis.colorFilter(imgHSV, trackBarPos)  
  
 imgCropped = imgMask[cropVals[1]:cropVals[2] + cropVals[1], cropVals[0]:cropVals[0] + cropVals[3]]  
 imgResult = imgResult[cropVals[1]:cropVals[2] + cropVals[1], cropVals[0]:cropVals[0] + cropVals[3]]  
 imgOpen = cv2.morphologyEx(imgCropped, cv2.MORPH\_OPEN, np.ones((5, 5), np.uint8))  
 imgClosed = cv2.morphologyEx(imgOpen, cv2.MORPH\_CLOSE, np.ones((10, 10), np.uint8))  
 imgFilter = cv2.bilateralFilter(imgClosed, 5, 75, 75)  
 imgContour, imgResult = utlis.getContours(imgFilter, imgResult)  
  
 ## TO DISPLAY  
 cv2.rectangle(img, (cropVals[0], cropVals[1]), (cropVals[0] + cropVals[3], cropVals[2] + cropVals[1]), (0, 255, 0), 2)  
 stackedImage = utlis.stackImages(0.7, ([img, imgMask, imgColorFilter], [imgCropped, imgContour, imgResult]))  
  
 # imgBlank = np.zeros((512, 512, 3), np.uint8)  
 # stackedImage = utlis.stackImages(0.7, ([img, imgBlank, imgBlank], [imgBlank, imgBlank, imgBlank]))  
  
 cv2.imshow('Stacked Images', stackedImage)  
  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
cap.release()  
cv2.destroyAllWindows()

import serial  
import numpy as np  
import cv2  
import math  
  
global ser  
  
  
def empty(a):  
 pass  
  
  
def initializeTrackBar():  
 cv2.namedWindow("HSV Value")  
 cv2.resizeWindow("HSV Value", 640, 240)  
 cv2.createTrackbar("HUE MIN", "HSV Value", 0, 179, empty)  
 cv2.createTrackbar("HUE MAX", "HSV Value", 26, 179, empty)  
 cv2.createTrackbar("SAT MIN", "HSV Value", 50, 255, empty)  
 cv2.createTrackbar("SAT MAX", "HSV Value", 255, 255, empty)  
 cv2.createTrackbar("VALUE MIN", "HSV Value", 0, 255, empty)  
 cv2.createTrackbar("VALUE MAX", "HSV Value", 255, 255, empty)  
  
  
def getTrackbarValues():  
 h\_min = cv2.getTrackbarPos("HUE MIN", "HSV Value")  
 h\_max = cv2.getTrackbarPos("HUE MAX", "HSV Value")  
 s\_min = cv2.getTrackbarPos("SAT MIN", "HSV Value")  
 s\_max = cv2.getTrackbarPos("SAT MAX", "HSV Value")  
 v\_min = cv2.getTrackbarPos("VALUE MIN", "HSV Value")  
 v\_max = cv2.getTrackbarPos("VALUE MAX", "HSV Value")  
 vals = h\_min, s\_min, v\_min, h\_max, s\_max, v\_max  
 return vals  
  
  
def connectToRobot(portNo):  
 global ser  
 try:  
 ser = serial.Serial(portNo, 9600)  
 print("Robot Connected ")  
 except:  
 print("Not Connected To Robot ")  
 pass  
  
  
def colorFilter(img, vals):  
 lower\_blue = np.array([vals[0], vals[1], vals[2]])  
 upper\_blue = np.array([vals[3], vals[4], vals[5]])  
 mask = cv2.inRange(img, lower\_blue, upper\_blue)  
 imgColorFilter = cv2.bitwise\_and(img, img, mask=mask)  
 ret, imgMask = cv2.threshold(mask, 127, 255, 0)  
 return imgMask, imgColorFilter  
  
  
def sendData(fingers):  
 string = "$" + str(int(fingers[0])) + str(int(fingers[1])) + str(int(fingers[2])) + str(int(fingers[3])) + str(  
 int(fingers[4]))  
 try:  
 ser.write(string.encode())  
 print(string)  
 except:  
 pass  
  
  
def getContours(imgCon, imgMatch):  
 global FingerCount  
 contours, hierarchy = cv2.findContours(imgCon, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)  
 imgCon = cv2.cvtColor(imgCon, cv2.COLOR\_GRAY2BGR)  
 bigCon = 0  
 myCounter = 0  
 myPos = np.zeros(4)  
 for cnt in contours:  
 area = cv2.contourArea(cnt)  
 if (area > 1000):  
 cv2.drawContours(imgCon, cnt, -1, (255, 0, 255), 3)  
 cv2.drawContours(imgMatch, cnt, -1, (255, 0, 255), 3)  
 peri = cv2.arcLength(cnt, True)  
 approx = cv2.approxPolyDP(cnt, 0.02 \* peri, True)  
 # APPROXIMATED BOUNDING BOX  
 x, y, w, h = cv2.boundingRect(approx)  
 ex = 10  
 cv2.rectangle(imgCon, (x - ex, y - ex), (x + w + ex, y + h + ex), (0, 255, 0), 5);  
 # CONVEX HULL &amp; CONVEXITY DEFECTS OF THE HULL  
 hull = cv2.convexHull(cnt, returnPoints=False)  
 defects = cv2.convexityDefects(cnt, hull)  
 bigCon += 1  
  
 for i in range(defects.shape[0]): # calculate the angle  
 s, e, f, d = defects[i][0]  
 start = tuple(cnt[s][0])  
 end = tuple(cnt[e][0])  
 far = tuple(cnt[f][0])  
 a = math.sqrt((end[0] - start[0]) \*\* 2 + (end[1] - start[1]) \*\* 2)  
 b = math.sqrt((far[0] - start[0]) \*\* 2 + (far[1] - start[1]) \*\* 2)  
 c = math.sqrt((end[0] - far[0]) \*\* 2 + (end[1] - far[1]) \*\* 2)  
 angle = math.acos((b \*\* 2 + c \*\* 2 - a \*\* 2) / (2 \* b \* c)) # cosine theorem  
 if angle <= math.pi // 1.7: # angle less than degree, treat as fingers  
 myPos[myCounter] = far[0]  
 myCounter += 1  
 cv2.circle(imgCon, far, 5, [0, 255, 0], -1)  
 cv2.circle(imgMatch, far, 5, [0, 255, 0], -1)  
  
 ## SENDING COMMANDS BASED ON FINGERS  
 if (myCounter == 4):sendData([1, 1, 1, 1, 1]);FingerCount = "Five"  
 elif (myCounter == 3):sendData([1, 1, 1, 1, 0]);FingerCount = "Four"  
 elif (myCounter == 2):sendData([0, 1, 1, 1, 0]);FingerCount = "Three"  
 elif (myCounter == 1):sendData([0, 0, 1, 1, 0]);FingerCount = "Two"  
 elif (myCounter == 0):  
 aspectRatio = w/h  
 if aspectRatio < 0.6:sendData([0, 0, 0, 1, 0]);FingerCount = "One"  
 else: sendData([0, 0, 0, 0, 0]);FingerCount = "Zero"  
 cv2.putText(imgMatch, FingerCount, (50, 50), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0, 0, 255), 2)  
 return imgCon, imgMatch  
  
  
def stackImages(scale, imgArray):  
 rows = len(imgArray)  
 cols = len(imgArray[0])  
 rowsAvailable = isinstance(imgArray[0], list)  
 width = imgArray[0][0].shape[1]  
 height = imgArray[0][0].shape[0]  
 if rowsAvailable:  
 for x in range(0, rows):  
 for y in range(0, cols):  
 if imgArray[x][y].shape[:2] == imgArray[0][0].shape[:2]:  
 imgArray[x][y] = cv2.resize(imgArray[x][y], (0, 0), None, scale, scale)  
 else:  
 imgArray[x][y] = cv2.resize(imgArray[x][y], (imgArray[0][0].shape[1], imgArray[0][0].shape[0]),  
 None, scale, scale)  
 if len(imgArray[x][y].shape) == 2: imgArray[x][y] = cv2.cvtColor(imgArray[x][y], cv2.COLOR\_GRAY2BGR)  
 imageBlank = np.zeros((height, width, 3), np.uint8)  
 hor = [imageBlank] \* rows  
 hor\_con = [imageBlank] \* rows  
 for x in range(0, rows):  
 hor[x] = np.hstack(imgArray[x])  
 ver = np.vstack(hor)  
 else:  
 for x in range(0, rows):  
 if imgArray[x].shape[:2] == imgArray[0].shape[:2]:  
 imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)  
 else:  
 imgArray[x] = cv2.resize(imgArray[x], (imgArray[0].shape[1], imgArray[0].shape[0]), None, scale, scale)  
 if len(imgArray[x].shape) == 2: imgArray[x] = cv2.cvtColor(imgArray[x], cv2.COLOR\_GRAY2BGR)  
 hor = np.hstack(imgArray)  
 ver = hor  
 return ver